Asian Journal of Control, Vol. 00, No. 0, pp. 1–10, Month 2011
Published online in Wiley InterScience (www.interscience.wiley.com) DOI: 10.1002/asjc.0000

On the use of feedback control in the design of computing system components

Martina Maggio, Alessandro Vittorio Papadopoulos, Alberto Leva

ABSTRACT

Feedback controllers are typically applied to computing systems by acting on some quantities ("tunable parameters" in the computer science lexicon, e.g., a packet drop rate) to attain some goal (e.g., a required bandwidth allocation). In other words, control loops are closed *around* the computing system. In the authors' opinion, this is definitely a partial use of control. Many critical parts of computing systems should not be *controlled*, but rather *re-designed in the form of controllers*. This paper formalises the statement above and illustrates some results, including application examples, to demonstrate the advantages of such a novel approach, and stimulate research on the matter.

Key Words: computing systems; discrete-time systems; PID control; predictive control.

I. Introduction

In these years, much attention is being devoted to "self-adaptive" [18] and "autonomic" [8] computing systems. Adopting the computer science jargon, such systems modify themselves so as to better respond to varying operating condition, such as workload [17], network throughput [19], services to offer [7], and so forth. The subject is apparently of interest also for the control community, where many of the involved adaptations have been formulated as modifications of tunable system parameters based on the desired and actual system behaviour—that is, feedback control problems [1] for which dynamic models in the system-theoretical sense can be applied [3].

However, observing the *control* literature on computing systems to date, a peculiar fact emerges. Typically, the controlled object is not the phenomenon of interest alone. Said phenomenon is often coupled with parts of the computing system that already act on it, and often attempt to solve – more or less empirically – some control-like problem. This system composed by phenomenon and control is usually taken as is and some loops are conveniently closed around it. To give a simple example, in several works the behaviour of a multitasking system is adjusted by adapting the parameters of an already installed scheduler, with feedback controllers [12]. In such cases, the phenomenon of interest is the behaviour of the scheduled tasks, but the loop is closed around that phenomenon *plus the existing scheduler* [13].

The excellent review [6], and more recent works such as [9, 18], confirm that the above way of using feedback control in computing systems – the "classical" one from now on – is definitely dominant. Nonetheless, in the opinion of the authors, such *modus operandi* has a very significant drawback. Typically, in fact, those parts of the system that *de facto* were already controlling the phenomenon of interest prior to the closure of the new feedback loop, were not conceived as dynamic systems, but directly as algorithms. Since the new loop has to include said parts, complex modelling paradigms most often come into play, and the overall design becomes difficult to carry out and assess.

In the long-term research to which this work belongs, a radically different approach is proposed: instead of taking the computing system *as is* and just

Manuscript received mm-dd-yyyy

The authors are with the Dipartimento di Elettronica e Informazione, Politecnico di Milano, Via Ponzio 34/5, 20133 Milano, Italy, e-mail `{maggio,papadopoulos,leva}@elet.polimi.it`; M. Maggio and A.V. Papadopoulos are PhD students at the Dipartimento di Elettronica e Informazione.

closing loops around it, *re-design* parts of that system in the form of one or more controllers, i.e., with system-theoretical methods. This requires to first evidence the phenomenon to be controlled (e.g., the concurrent execution of a pool of tasks) and describe it in a system-theoretical framework, obtaining a model of the controlled system in open loop (the "plant" from now on). Note that such an object is hardly ever mentioned in the classical literature.

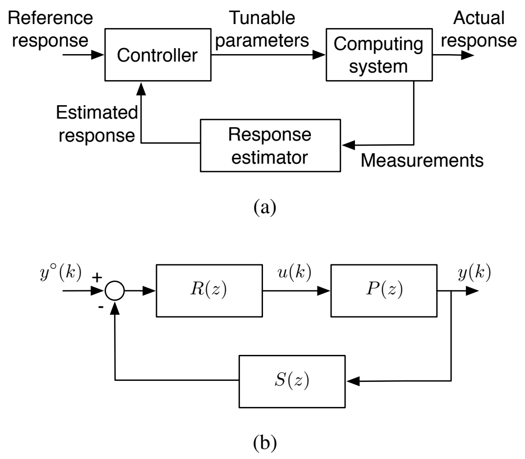
Once the core phenomenon is isolated, there is little or no need to include in the model any entity that is not keen to be described by a dynamic system (the following examples will clarify this). Hence, simple modelling paradigms can be used, allowing for powerful synthesis and analysis tools.

Having characterised the plant, a controller (model) is then designed, generally with standard methods—and not surprisingly, leading to simpler and computationally lighter solutions than with the classical approach. The control algorithm finally arises in a natural manner, as the implementation of the so obtained controller model.

This paper, extending the results of [14, 16], first illustrates the new proposed way of using control in computing systems, by applying it to task scheduling. Modelling the core phenomenon leads to conclude that all the major existing scheduling policies are particular cases of a single discrete-time dynamic system. The adoption of such a unitary framework makes it straightforward to apply solid and simple control techniques, namely PID and predictive control, and to formally assess the results. Some words are also spent on the generalisation of the approach and on its application to other problems, thereby sketching out future research.

II. Motivation and novelty

A representative example of the classical approach can be found in [6]. The typical control scheme proposed therein looks like figure 1a. Taking this viewpoint, the "computing system" block represents an already functional object, and the controller just tunes some of its parameters. However, two are the main problems with this approach. One is that modelling the computing system taken *as is* is complex. The other is that the tunable parameters are not necessarily the best inputs to control the phenomenon of interest. Further, when dealing with modelling and control design, the scheme in figure 1a must be translated into something like figure 1b. This is far from trivial, and typically goes through black box identification or similar techniques.



(a)

(b)

Figure 1. Typical control schemes of the classical approach.

On the other hand, the authors' viewpoint is that if one implements part of a system (as software) and then for its control needs to identify a black box model for it, something must be wrong. The approach used here in fact does not start from a functional system, but only from the physical phenomena that decide its behaviour. The result is a system *part of which* is a controller, or in other words, where removing the controller would not result in a still functional system. The approach strength comes from observing that in several cases (scheduling included) modelling the core phenomena only is extremely simple.

As a result, with this approach, the transition from figure 1a to figure 1b is totally straightforward, or better, the models of the controlled and the control system already start out in a form compatible with figure 1b. The advantages in terms of clarity and analysis possibilities should be apparent, and are now shown by addressing the scheduling problem.

III. A preemptive scheduler in the form of a feedback controller

3.1. Modelling the core phenomenon to control

Consider a single-processor multitasking system with a preemptive scheduler. Let N be the number of tasks to schedule, assumed constant (task arrival and termination is simply a matter of reinitialisation). Define the "round" as the time between two subsequent scheduler interventions. In the following, vector $\tau_p(k) \in \Re^N$ represents the CPU times *actually* allocated to the tasks in the k-th scheduling round, and $\tau_r(k) \in \Re$ is the *actual* time duration of the k-th round; $\rho_p(k) \in \Re^N$

contains the “times to completion” (i.e., the remaining CPU time needed by the task to end its job) at the beginning of the k-th round for the tasks that have a duration assigned (elements corresponding to tasks without an assigned duration will be $+\infty$), while $b(k) \in \Re^{n(k)}$ and $\delta b(k) \in \Re^{n(k)}$ contain the “bursts”, i.e., the CPU times allotted by the scheduler to the tasks at the beginning of the k-th round, and the disturbances possibly acting on the scheduling action during the k-th round (for example because some tasks release the CPU before their bursts elapse, because of an interrupt management, and so forth); $n(k)$ $(1 \leq n(k) \leq N \forall k)$ is the number of tasks that the scheduler considers at each round, being it constant and equal to one in the traditional policies. Denote by t the total time actually elapsed from the system initialisation. With the definitions above, a simple plant model, sufficient however for the purpose of this work, is

$$\left\{ \begin{array}{rcl} \tau_p(k) & = & S_\sigma b(k-1) + \delta b(k-1) \\ \tau_r(k) & = & r_1 \tau_p(k-1) \\ \rho_p(k) & = & \max\big(\rho_p(k-1) - S_\sigma b(k-1) \\ & & -\delta b(k-1), 0\big) \\ t(k) & = & t(k-1) + \tau_r(k) \end{array} \right. \tag{1}$$

where r_1 is a row vector of length N with unit elements, and $S_\sigma \in \Sigma$ a $N \times n(k)$ switching matrix. The elements of S_σ are zero or one, and each column contains at most one element equal to one. Matrix S_σ therefore determines which tasks are considered in the particular round at hand, to the advantage of generality (and possibly for multiprocessor extensions). Notice that, since $n(k)$ is bounded, set Σ is finite for any N.

Moreover, the τ_p, τ_r and t equations in (1) form a linear, switching discrete-time dynamic system, apart from the obvious input saturation constraint given by the impossibility of negative bursts and durations. Such control saturations are managed with standard antiwindup. The ρ_p equation is conversely nonlinear. However, given the role of disturbances in the adopted framework, a task that terminates before exhausting its burst is simply modelled with a negative disturbance element on that burst, and then removed from the pool. Since the relevant fact is here that disturbances are exogenous to the scheduler only, one can therefore disregard the nonlinear equation as not relevant for the problem and safely treat the model as the linear switching one

$$\left\{ \begin{array}{rcl} \tau_p(k) & = & S_\sigma b(k-1) + \delta b(k-1) \\ \tau_r(k) & = & r_1 \tau_p(k-1) \\ t(k) & = & t(k-1) + \tau_r(k) \\ \rho_p(k) & = & \rho_p(k-1) - S_\sigma b(k-1) - \delta b(k-1) \end{array} \right. \tag{2}$$

3.2. Controlling the core phenomenon

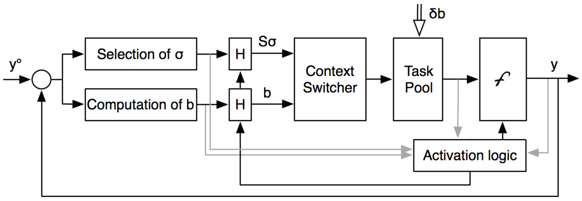


Figure 2. The proposed approach applied to scheduling.

In figure 2 a general scheme is presented for the proposed approach applied to scheduling. The plant (or core phenomenon) is composed of just the task pool and the disturbances possibly acting on it, while the controller is naturally partitioned into three entities. The first one is a block that selects the switching signal $\sigma(k)$, thus yielding $S_\sigma(k)$. The second one is a block that computes the burst vector $b(k)$. The third one is the activation logic block, that determines when the scheduler has to intervene according to the control policy. Notice that in the most general case the activation logic component receives as inputs the signals $b(k)$ and $S_\sigma(k)$ computed by the each other block, as well as measures of the plant output and of the current cost function f, that can be arbitrarily chosen. For example, one could decide to maintain the bursts constants for more than one round, or have the scheduler act only when some threshold is exceeded by some measured quantity related to the system performance (i.e., memory usage), and so on. In this work the activation logic is not addressed, and was mentioned here only to stress the generality of the used formalism. Notice however that the proposed formalism makes that logic transparent to the controller design.

Setting up the scheme of figure 2, a relevant merit of the proposed approach is revealed. Modelling the core phenomenon naturally leads to evidence – as its states (and outputs) – the basic elements for virtually any performance metrics one may conceive, thus to formulate specifications on those metrics as Set Point (SP) following requests. In fact, any possible scheduling objective can be formulated as a cost function and/or a set of input and/or state constraints for (2). In other words, any objective can be expressed by combining a desired behaviour $\tau_p^\circ(k)$, a set of input constraints in the form $b(k) \in B(k)$, and a cost function in the form $J(\phi, \tau_p^K, b^K, \rho_p^K)$, where $\tau_p^K := [\tau_p(0)\, \tau_p(1) \ldots \tau_p(K)]$, $b^K := [b(0)\, b(1) \ldots b(K)]$, $\rho_p^K := [\rho_p(0)\, \rho_p(1) \ldots \rho_p(K)]$, and ϕ is a convenient vector of parameters. This is another significant novelty with respect to the mainstream computer science literature, that concentrates mainly (not to say exclusively) on *off-line* schedulability analysis, especially for real-time systems, with the aim of verifying the *attainability* of an objective; the problem of driving the *on-line* behaviour of the scheduler is seldom addressed.

Another important remark is that the formalism can describe virtually any existing scheduling policy, by merely imposing conditions on n and/or S_σ. For example

- $n = 1$ and a N-periodic S_σ with
  $$S_\sigma(k) \neq S_\sigma(k-1), 2 \leq k \mod N \qquad (3)$$
  produce all the possible Round Robin (RR) policies having the (scalar) $b(k)$ as the only control input, and obviously the pure round robin if $b(k)$ is kept constant,
- generalisations of the RR policy are obtained if the period of S_σ is *greater* than N, and (3) is obviously released,
- $n = 1$ and a S_σ chosen so as to assign the CPU to the task with the minimum row index and a ρ_p greater than zero produces the First Come First Served (FCFS) policy,
- $n = 1$ and a S_σ selecting the task with the minimum ρ_p yields the Shortest Remaining Time First (SRTF) policy,
- $n = 1$ and a S_σ that switches according to the increasing order of the initial ρ_p vector produces the Shortest Job First (SJF) policy.

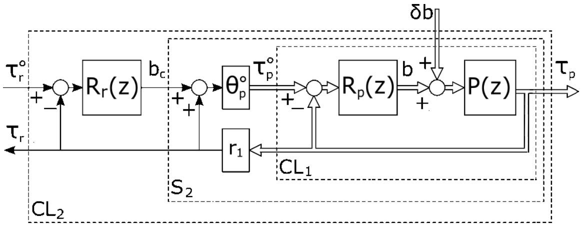


Figure 3. The synthesised “scheduler as controller” scheme.

However, the main interest of the proposed approach is that it allows to devise *new* policies *in a system-theoretical manner*. A quite general way to do so leads to the cascade control system of figure 3, where τ_r° is the required scheduling round duration, and

$$\theta_p^\circ \in \Re^N, \qquad \theta_{p,i}^\circ \geq 0, \quad \sum_{i=1}^{N} \theta_{p,i}^\circ = 1 \qquad (4)$$

is the vector containing the required CPU time fractions to be allotted to each task. The R_p regulator aims at achieving said CPU time distribution within the round, while the R_r regulator leads the system to attain the desired round duration. As for the control specifications' interpretation, R_p takes care of distributing the CPU for fairness, or to have each task accumulate enough time to accomplish its workload within prescribed deadlines, or any combination hereof. At the same time, R_r makes the scheduler regain control as frequently as desired, thereby providing a prescribed degree of system responsiveness.

3.3. Applying standard control techniques

This section shows how the scheme of figure 3, and more in general the proposed approach, are easily turned into functional solutions by bringing in well assessed control design methods. Tests and discussions follow in the next section.

3.3.1. PI control

By choosing R_p as a diagonal integral regulator with gain k_{pi}, one makes the closed-loop system denoted by CL_1 in figure 3 a diagonal one the eigenvalues of which are the pair $0.5 \mp \sqrt{0.25 - k_{pi}}$, with multiplicity N. If τ_p° were chosen as $\theta_p^\circ \tau_r^\circ$, then CL_1 would control both the CPU distribution and the round duration, but there would be two problems. First, the dynamics of those two controls would be ruled by the same eigenvalues, which can be inadequate in some cases. For example, one may want the CPU distribution to move smoothly from one situation to another, but the round duration to respond very quickly to its SP. Second, and more serious, should some task be blocked, the round duration SP could not be attained.

Consider therefore the system denoted by S_2 in figure 3. With the chosen R_p, its eigenvalues are 0 and 1, with multiplicity 1 each, and the pair $0.5 \mp \sqrt{0.25 - k_{pi}}$, with multiplicity $N - 1$. Notice that said eigenvalues do not depend on θ_p°. The single-input, single-output system with input b_c and output τ_r seen by R_r in figure 3 has thus the transfer function

$$\frac{T_r(z)}{B_c(z)} = \frac{k_{pi}}{z(z-1)}. \qquad (5)$$

Coming to $R_r(z)$, its role is to introduce an additive correction b_c (named “burst correction” for apparent reasons) to the bursts computed by R_p. The simplest idea is to choose $R_r(z)$ as a purely proportional controller, i.e., $R_r(z) = k_{rp}$. In this case the eigenvalues of system CL_2 in figure 3 are those of S_2 with pair (0,1) replaced by $0.5 \mp \sqrt{0.25 - k_{pi}k_{rp}}$. This choice will be termed from now on "I+P".

In the case of a constant required CPU distribution, the I+P scheme can assign the dynamics of τ_p and τ_r in a (partially) independent manner. If, conversely, a variable CPU distribution is to be considered, the same scheme can be viewed as a linear switching system with switch signal θ_p°. However, its eigenvalues do not depend on the switching signal, and in force of well known results, see e.g. [4], there surely exists a finite dwell time ensuring the exponential stability of the scheme as switching system. Details on the matter would stray from this work, but still notice the analysis possibilities yielded by the proposed approach. The interested reader can refer to works such as [5,20].

The I+P is computationally very light, and with it the closed-loop transfer function from τ_r° to τ_r is

$$\frac{T_r(z)}{T_r^\circ(z)} = \frac{k_{pi}k_{rp}}{z^2 - z + k_{pi}k_{rp}} \tag{6}$$

thus allowing for a simple choice of the parameters k_{pi} and k_{rp}. The (only) drawback that the round duration control is lost if a task stays blocked. If one cannot guarantee that no persistent task blockings arise, it is advisable to select for R_r a PI structure, i.e., $R_r(z) = k_{rr}(z - z_{rr})/(z-1)$, leading to what in the following will be termed "I+PI". In this case the $2N+1$ eigenvalues of CL_2 have a long expression omitted for brevity, but still do not depend on the switching signal θ_p°. Hence, the same stability considerations above apply. With I+PI, the closed-loop transfer function from τ_r° to τ_r is

$$\frac{T_r(z)}{T_r^\circ(z)} = \frac{k_{pi}k_{rr}(z - z_{rr})}{z^3 - 2z^2 + (1 + k_{pi}k_{rr})z - k_{pi}k_{rr}z_{rr}} \tag{7}$$

and the choice of the parameters (k_{pi}, k_{rr}, and z_{rr}) is just slightly more articulated than it is with I+P, while also the computational burden is only a bit higher. A discussion on the time complexity of policies like those introduced, in comparison with more traditional ones, can be found in [11].

3.3.2. Predictive control

In this section, R_r in figure 3 is replaced by a RHPC (Receding Horizon Predictive Controller), while R_p is maintained. In figure 3 with the inner loops closed. If all the I gains are equal and no task is blocked, the transfer function from $B_c(z)$ to $T_r(z)$ is given by (5). Suppose now that said gains are not equal anymore: expressing the i-th gain as $\beta_i k_{pi}$, and representing a blocked task by zeroing its gain – i.e., replacing the first equation of (2) with $\tau_p(k) = \Gamma b(k-1)$ where $\Gamma := diag\{\gamma_i\}$ is a matrix with one or zero diagonal elements (zero means a blocked task) – the same transfer function becomes

$$\frac{T_r(z)}{B_c(z)} = \frac{k_{pi}}{z(z-1)} \frac{\Delta_N(z)}{\Delta_D(z)} \tag{8}$$

where $\Delta_N(z)$ and $\Delta_D(z)$ are polynomials in z of degree N, the expression of which is lengthy and thus omitted. Suffice to say that $\Delta_N(z)/\Delta_D(z)$ can alter the dynamics of (8) significantly with respect to (5). Suppose then to select as R_p a diagonal integral regulator with *different* gains expressed for convenience as $\beta_i k_{pi}$, which corresponds in the state-space to

$$\begin{aligned} A_{R_p} &= C_{R_p} = I_{N\times N} \\ B_{R_p} &= k_{pi}[\beta_1\ \beta_2\ \ldots\ \beta_N] I_{N\times N} \\ D_{R_p} &= 0_{N\times N} \end{aligned} \tag{9}$$

where $I_{N\times N}$ and $0_{N\times N}$ are respectively the identity and the zero matrix of dimensions $N \times N$. Introducing the information of blocked tasks, the system denoted in figure 3 by S_2 has the switching dynamic matrix

$$A_{S_2} = \begin{bmatrix} 0_{N\times N} & B_\Gamma C_{R_p} \\ B_{R_p}(\theta_p^\circ r_1 - I_{N\times N}) & A_{R_p} \end{bmatrix} \tag{10}$$

where

$$B_\Gamma = \gamma I_{N\times N} = \mathrm{diag}\{\gamma_1, \gamma_2, \cdots, \gamma_N\} \tag{11}$$

and $\gamma_j = 1$ if the task is active and $\gamma_j = 0$ if the task is blocked (for example waiting for a resource to be freed). Based on the state-space model just sketched, a simple RHPC control law can be designed as

$$b_c(k) = b_c(k-1) + \Delta b_c(k) \tag{12}$$

where $\Delta b_c(k)$ is computed as

$$\Delta b_c = \overbrace{[1\ 0\ \ldots\ 0]}^{N_c}(\Phi'\Phi + \bar{R})^{-1}\Phi'\,(\bar{T}_r^\circ \tau_r^\circ(k) - Fx(k)) \tag{13}$$

where N_c is the control horizon, and Φ and F depend on the model matrices in a well established manner, see e.g. [2], $\bar{R}$ being a diagonal matrix in the form that $\bar{R} = wI_{N_c\times N_c}(w \geq 0)$ where w is used as a tuning parameter for the desired closed-loop performance. Finally, $\bar{T}_r^\circ$ is an all-ones column vector of length N_p (prediction horizon). Of course, in the case of "symmetric desires", where the integral gains are equal, and if no information about the blocked tasks is available (or equivalently, the tasks are always considered not to be blocked), the resulting SISO system with input b_c and output τ_r has the transfer function (5), and it can be verified that the so obtained control is equivalent to I+P.

Far more interesting are the *additional* possibilities of the RHPC, in the case of "asymmetric" desires on the scheduled tasks. Tasks running on a generic system have in fact very different purposes, requirements, and behaviours. For example, in the mainstream computer science literature, they may be classified as "I/O-bound" or "CPU-bound". The former type makes heavy use of I/O devices and spends much time waiting for I/O operations to complete; the latter type are number-crunching applications that require a lot of CPU time. Another classification distinguishes three classes of tasks: the interactive ones (tasks which interact constantly with their users, and therefore spend a lot of time waiting for key-presses and mouse operations), batch ones (tasks which do not need user interaction, and hence often run in the background) and real-time tasks (tasks which should never be blocked by lower-priority tasks, have a short response time and, most important, exhibit for such response as low a variance as possible). Whatever is the case, different kind of tasks which must run on the same computing system are treated as different entities. Again, this shows that the main advantage of the proposed "fully control theoretical" standpoint, is that such a classification does not lead to different scheduling structures, but only to particular cases of a single one.

IV. Examples

4.1. I+P and I+PI

This section presents some results with I+P and I+PI, and different parameter choices. The applied *stimuli* are (a) some modifications of the required CPU distribution, to show that the basic goal of the scheduler is attained, (b) some modifications to the pool of tasks, to prove that re-initialising the controller is straightforward, (c) some task blockings, to verify the correct treatment of that case by the I+PI solution, and (d) some modifications of the desired round duration, to illustrate that the system responsiveness can actually be changed on-line.

Both I+P and I+PI require the specification of k_{pi}. The simplest and somehow "standard" choice is to take $k_{pi} = 0.25$, which sets the eigenvalues of CL_2 in figure 3 to be 0, 1 (with multiplicity 1 each) and 0.5 (with multiplicity $2N-2$). This will be the choice in all the tests that follow. With I+P, the only further parameter to set is k_{rp}. Again, one can select a value based on the position of the eigenvalues of CL_2, and along this reasoning another "standard" choice can be $k_{rp} = 1$.

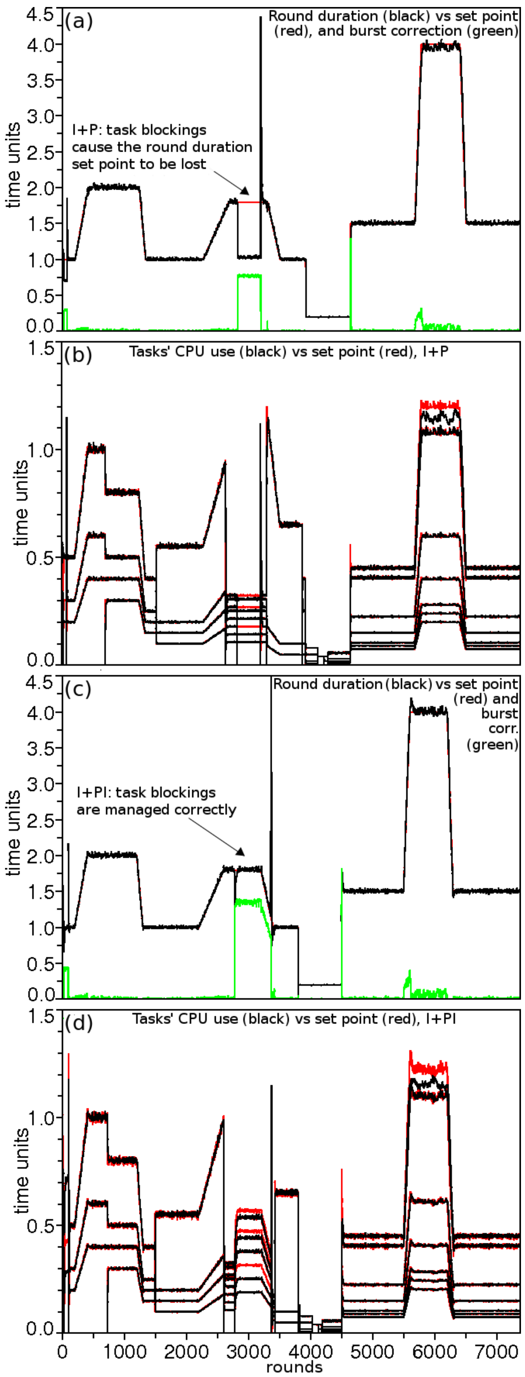


Figure 4. Test results with I+P (a,b) and I+PI (c,d) control.

Figure 4 reports the round duration τ_r versus its SP τ_r° and the additive correction b_c exerted by R_r with I+P (a) and I+PI (c), and the tasks' CPU use τ_p versus their SP, again with I+P (b) and I+PI (d). The abscissæ axes are graduated in time units (not rounds), which by the way allows to see that in all the examples the same number of rounds actually lasts (more or less) the same time. The test also comprises some task blockings. More precisely, task T_2 is blocked from round 30 to round 100, while between rounds 2100 and 2450 a blocking occurs for T_3, T_6, and T_8. This is quite severe a perturbation, owing to the long blocking durations. Apparently I+P works satisfactorily, but in the case of (persistent enough) blockings is not capable of attaining the required round duration. On the contrary, I+PI manages blockings correctly.

One final consideration is useful. In the mainstream scheduling literature, it is often claimed that policies should be *adaptive*, which is obtained by changing the *parameters* of some (feedback) control law. In this work, control parameters are set on more a model-free basis, in fact *totally* model-free with I+P(I), and never changed. This permits to "tune once and then touch only SPs", which is apparently in favour of a system-theoretical view on the problem.

4.2. RHPC

In the previous section, a symmetric internal loop CL_1 was considered (see figure 3), making the assumption that all the tasks can be treated in the same manner. A more general approach is to use different weights for the different tasks' I controllers in the internal loop, in order to distinguish the low-priority tasks from the high-priority ones by means of lower or higher gains respectively, leading to different response times.

The I+PI approach can still lead to good results, from the above point of view, as exemplified in figure 5, top, that is organised in the same way as plots (a,c) in figure 4. However, when the internal loop has different regulators its design is not so simple as it is in the symmetric case, and in particular, it is not independent of θ_p° anymore. With the RHPC it is on the contrary quite straightforward to design the controller by resorting to (10) and merely releasing the symmetry assumption. Analysing the corresponding plots of figure 5, bottom, one can see how the predictive approach leads to substantially analogous results with respect to I+PI, although its design is simpler.

Both with the PI and with the RHPC, we have reached the goal of rejecting such disturbances and of following the SP signal. However, there is an open *actuation* problem: what happens to the burst assigned to each task when one or more tasks are blocked? How should the actuator distribute the CPU time among the tasks within the round?

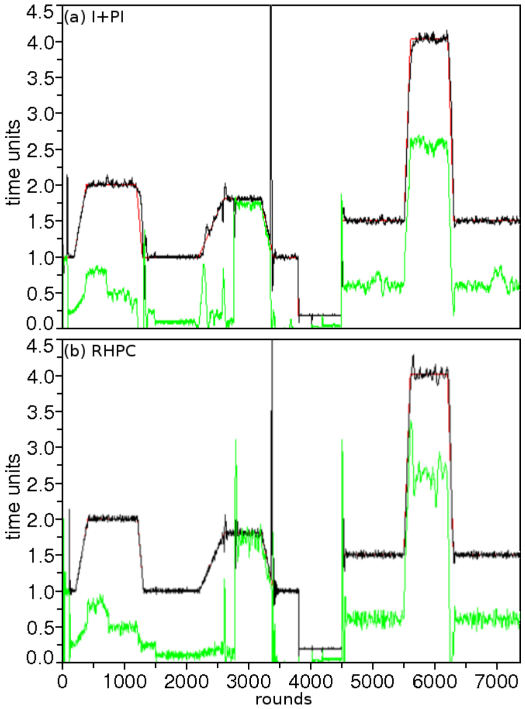


Figure 5. I+PI (a) versus RHPC (b) with different gains in the internal loop: both plots are organised as figure 4(a,c).

With I+PI, there are two main solutions of this actuation problem. The first solution is to maintain the round duration constant, as if all tasks were scheduled, and the time that should have been allotted to the blocked tasks were considered "idle time". This idea implies an action on the actuators, not on the controller, in that if a task returns the CPU before the expiration of its burst, the system (more precisely, its actuating part) just has to wait till the allocated burst is exhausted. The second solution is to re-distribute the time of the round duration among the active non blocked tasks. As in the first case, the round duration remains constant, but the percentage of CPU time of each task changes. The use of the RHPC implies new possibilities in this *panorama*. For instance, one can open the external loop, releasing the round duration control, which may change at each scheduling round. The round duration becomes the sum of the bursts assigned to the tasks at the k-th round, except that assigned to the blocked tasks. The information about blocked tasks at the last step can be used by RHPC to predict the output.

This solution is much better than those offered by the simple PI control structure. For example, in the case of a CPU for a desktop PC, it may not be so dramatic if the CPU is idle because of a blocked task, but in the case of a web server or a router, where the waste of CPU time is a greater damage, it would be better to use all the available CPU time or bandwidth, or, in general, all the available resources. Also, with RHPC, it is reasonably simple to account for task blockings by simply having them detected by the operating system (details on that would stray from this work but pose non serious technical problems), and change the predictive model used in the controller by acting on (11). The so obtained results are shown in figure 6.

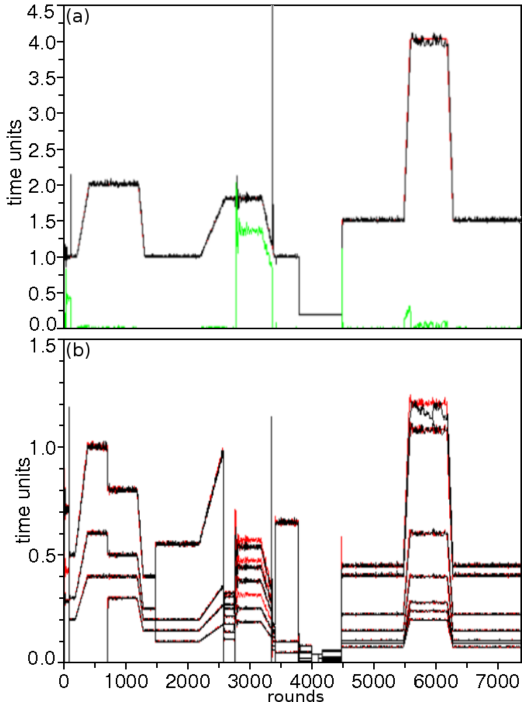


Figure 6. Results with RHPC in the presence of task blockings: plots (a) and (b) are respectively organised as figure 4(a,b).

4.3. Discussion

Along the proposed approach, two well established control techniques were applied to task scheduling, in the presence of “symmetric” and “asymmetric” desires, and possible task blockings. With PI-based control, namely I+PI, the blocked tasks management corresponds to the switching of the system, and does not impact on the round duration which remains the same, while the CPU time assigned to each task may remain the same (with some idle time corresponding to the blocked tasks) or may be re-distributed among non blocked tasks (increasing each CPU time percentage). A predictive control approach can lead to better solutions for the blocking tasks problem. Moreover, I+PI is not *scalable*, in the sense that if we decide to use different weights on each computed error, the model becomes much more complicated and it is no longer reducible to a second order model, i.e., denoting by N the number of the running tasks, in the worst case (a different weight for each task) the model becomes of order $2N$. Thus, in this case, it is convenient to use a more complex control technique, which allows to specify different weights, with the same computational complexity. To summarise the reported comparison of I+PI and RHPC, one can state that the former is tendentiously simpler to set up but less scalable. As such, I+PI is a good solution for “symmetric” problems, i.e., where the error weights can be taken equal. On the other hand, in situations characterised by asymmetry on the desires, it is more convenient to use predictive schemes such as the RHPC, that have greater computational complexity, but succeed in a wider variety of situations.

Comparative experiments were also conducted with the Hartstone PH (Periodic Processes, Harmonic Frequencies) series benchmark [21], the *rationale* of which is to increase the load of a system until a task deadline miss is observed. Each of the four benchmark stresses the system in different ways, see [21] for full details. Table 1 shows the number of iterations (i.e., system load increases) before the first miss (the higher, the better). Also, the number of context switches in the test iteration before that with the first miss is shown in parentheses. The I+PI either achieves better results with respect to the RR, or comparable ones with less context switches. This test is conceived for EDF to perform better than every other algorithm (since the system is not overloaded), therefore EDF data are reported as the upper bound. Recall however that EDF inherently relies on the concept of task deadline, thus it provides an upper performance bound only for a *schedulable* pool of periodic tasks (as the Hartstone benchmark). The proposed approach is conversely general, hence suitable for a pool composed of tasks of any type. Moreover, as shown e.g. in [15] and contrary to EDF, it can also cope successfully with (transient) CPU over-utilisation.

V. The proposed approach in general

The scheduling treatise shown so far is just an example of what can be done if one accepts to somehow reverse the mainstream perspective on the use of control techniques in computing systems, i.e., to *design systems as controllers* instead of *controlling systems designed as algorithms*. Of course no claim is here made that the proposed models can represent computing

	Benchmark No.	I		II		III		IV	
Policy	EDF	31	(10785)	16	(3487)	8	(703)	8	(959)
	RR	6	(2705)	15	(11051)	4	(1858)	7	(4364)
	I+PI	27	(193109)	15	(10835)	8	(6270)	8	(8287)

Table 1. Hartstone PH series benchmark results with different scheduling policies (Earliest Deadline First, Round Robin, and I+PI).

systems in general, but several significant problems in that domain can be addressed. Hence, in the authors' opinion, said perspective reversal opens a wealth of possibilities. To sketch out the *panorama*, it is useful to abstract the general key facts of the proposed approach. First, for each system adaptation capability (in the computer science meaning) a controller is designed, and for each control design, care is taken to isolate the core phenomenon, accepting to undergo some *plant* re-design. Then, the adopted modelling formalism is confined to discrete-time dynamic systems. Finally, the time index counts the interventions of the controller, i.e., the obtained controls are discrete-time but not sampled-signals ones. If the specifications include time-related quantities, they are treated as state variables (see e.g. τ_p and τ_r in this work).

Three other characteristics of computing systems make the proposed approach particularly effective. First, variables are often measured without (or with negligible) errors, which is far from true in virtually any other case. Here too, the approach naturally leads to model uncontrollable actions on the system as disturbances—an attitude that the reported examples have shown to be effective in practice. Second, when two or more controls interact, this almost invariantly happens on a time scale that is far slower than that of all those controls, which allows to address the individual adaptation functionalities with the reasonable certainty that assembling them together in essentially a decentralised manner will lead to a functional system. Third, the task variabilities typically encountered are either very slow, or abrupt but very sporadic (like e.g. a drop in a server reliability), permitting the successful use of quite simple adaptive techniques.

Finally, the proposed approach offers different potentialities if compared to classical techniques adopted in the computer science community [10]. Not only static properties on the system (typical of the computer science analysis), but also dynamic ones (typical of the control theory domain) can be formally proven, e.g. settling time, disturbance rejection, proof of stability, and so on. Apparently enough, doing so improve the performance of the controlled system since those aspects are taken into account yet in the design phase of the controller (in this case the scheduler in an Operating System). This was not possible when the scheduler was designed directly as an algorithm.

VI. Conclusions and future work

The main proposal of the research to which this paper belongs is to employ control-theoretical methods and tools not to *control*, but to *design* computing system components. A representative example of the approach was here presented, namely the "design as a controller" – i.e., the design using system theoretic methods – of a preemptive scheduler for a uniprocessor multitasking system, with two different control techniques applied to the same general scheme, and well qualified so that the one best suited for the problem at hand can be selected. The approach advantages were evidenced in terms of controllability and interpretation of the involved quantities and parameters. Moreover, it was pointed out that when the problem is formulated on an equation-based model for which a controller is designed, the resulting system can be analysed in terms of standard indicators like steady-state error and settling time, to the advantage of a well grounded assessment without the need for extensive testing.

The presented research opens more than one future perspective. Sticking to scheduling, for example, very interactive tasks can produce and receive a lot of interrupts. Another scheduling level could be added to better manage such situations. Also, extensions to multicore environments can be envisaged. A solution may select the core devoted to the execution of each tasks and run a controller per core; however, one could also aim at balancing the core workloads. Another interesting topic is the definition of SP profiles. If information about the task pool is available, such as deadlines or expected resource usages, these information could be exploited for SP generation. For example, ramp-like SPs could be generated for the accumulated CPU time, where the ramp slope for each task depends on its importance. Also, one can introduce different metrics, for example filtering or averaging the affected quantities (such as the CPU distribution "fairness") over conveniently chosen time horizons, and so forth.

Finally, the relationship between the performance indices derived from the computing system domain and the one coming from the control-theoretical one are of interest. In the former case, for example, a web server is evaluated in terms of QoS indicators. In the latter one, instead, a controller is usually evaluated in terms of stability, accuracy (steady-state error) and transient performance (settling time and overshooting). The relationship between these two approaches deserves further investigations. On a similar front, since "redesigning as controllers" is powerful but may impact system design significantly, it will be necessary to study how to port "control theoretical" solutions onto systems that were conceived and are organised in hardly compatible a way. For example, the memory space of a task in a priority-based scheduler holds a lot of data that are useless for I+PI. It is however unrealistic that the API exposed by the scheduler requires application designers to significantly modify their code, whence the necessity for a number of software engineering studies in coordination with the addressed system-theoretical issues.

References

1. K.J. Åström and B. Wittenmark. *Adaptive Control*. Addison-Wesley, Boston, MA, 1995.
2. E.F. Camacho and C. Bordons. *Model predictive control*. Springer Verlag, London, 2004.
3. Hua Fan, Yong Ren, and Xiuming Shan. Estimation of the variances of tcp/red using stochastic differential equation. *Asian Journal of Control*, pages n/a–n/a, 2011.
4. J.C. Geromel and P. Colaneri. Stability and stabilization of discrete time switched systems. *Int. Journal of Control*, 79(7):719–728, 2006.
5. Rongwei Guo and Yuzhen Wang. Stability analysis for a class of switched linear systems. *Asian Journal of Control*, pages n/a–n/a, 2011.
6. J. Hellerstein, Y. Diao, S. Parekh, and D.M. Tilbury. *Feedback Control of Computing Systems*. Wiley, Hoboken, NJ, 2004.
7. J. Hellerstein, Y. Diao, S. Parekh, and D.M. Tilbury. Control engineering for computing systems, industry experience and research challenges. *Control Systems, IEEE*, 25(6):56–68, Dec 2005.
8. M.C. Huebscher and J.A. McCann. A survey of autonomic computing—degrees, models, and applications. *ACM Comput. Surv.*, 40:1–28, 2008.
9. E. Kalyvianaki, T. Charalambous, and S. Hand. Self-adaptive and self-configured CPU resource provisioning for virtualized servers using kalman filters. In *Proc. 6th int. conf. on Autonomic computing*, ICAC '09, pages 117–126, Barcelona, Spain, 2009. ACM.
10. E. Kinber and C. Smith. *Theory of Computing: A Gentle Introduction*. Prentice Hall, Boston, MA, 2001.
11. A. Leva and M. Maggio. Feedback process scheduling with simple discrete-time control structures. *Control Theory Applications, IET*, 4(11):2331–2342, 2010.
12. C. Lu, J.A. Stankovic, S.H. Son, and G. Tao. Feedback control real-time scheduling: Framework, modeling, and algorithms. *Real-Time Syst.*, 23:85–126, 2002.
13. M. Maggio and A. Leva. A new perspective proposal for preemptive feedback scheduling. *Int. Journal of Innovative Computing, Information and Control*, 6(10):4363–4377, 2010.
14. M. Maggio and A. Leva. Toward a deeper use of feedback control in the design of critical computing system components. In *Proc. 49th Conf. on Decision and Control*, pages 5985–5990, Atlanta, GA, 2010.
15. M. Maggio, F. Terraneo, and A. Leva. Implementation and evaluation of a control-theoretical schedule. *ICIC Express Letters*, 4(6(B)):2343–2347, 2010.
16. A.V. Papadopoulos, M. Maggio, S. Negro, and A. Leva. Enhancing feedback process scheduling via a predictive control approach. In *18th IFAC WC*, pages 13522–13527, Milan, Italy, 2011.
17. A. Robertsson, B. Wittenmark, M. Kihl, and M. Andersson. Design and evaluation of load control in web server systems. In *Proc. 2004 American Control Conference*, volume 3, pages 1980–1985, Boston, MA, 2004.
18. M. Salehie and L. Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.*, 4:1–42, 2009.
19. L. Sha, X. Liu, Y. Lu, and T.F. Abdelzaher. Queuing model based network server performance control. In *23rd IEEE Real-Time Systems Symposium*, pages 75–80, Austin, TX, Dec 2002.
20. Limin Wang, Cheng Shao, and Xuanyu Liu. State feedback control for uncertain switched systems with interval time-varying delay. *Asian Journal of Control*, pages n/a–n/a, 2010.
21. N.H. Weiderman and N.I. Kamenoff. Hartstone uniprocessor benchmark: Definitions and experiments for real-time systems. *Real-Time Systems*, 4:353–382, 1992. 10.1007/BF00355299.